# 球面距离计算方法及精度比较*

樊东卫，何勃亮，李长华，韩军，许允飞，崔辰州

（中国科学院国家天文台，北京100101）

**摘 要：**球面距离（角间距）计算是天文或地理学中极常用的计算之一，也是目标查找、锥形检索、交叉证认等方法的基础。数学上，通过球面几何可以直接计算出两点的距离，前人也已经推导出了多个复杂程度不一进行计算。但是由于计算机的精度有限，在进行数值计算时有舍入误差，导致公式计算结果出现偏差。对几个常用的球面距离计算公式进行了考察，测试并对比它们在不同计算环境下的精度与优缺点。此外还展示并比较了几种常用天文软件包、数据库的球面距离计算方法，以期有助于天文工作者选择适合自己当前需要的计算方法。

**关键词：** 数值计算；球面距离；角间距；计算精度

**中图分类号：**P121　　**文献标识码**：A　**文章编号：**1672-7673(2019)



平面距离只需要计算连接两点线段的长度，通过二维直角坐标使用勾股定理可以很容易地计算出来。由于只涉及乘法、加法与开平方运算，在计算机中也可以保留很高的精度。而球面距离计算则要复杂得多。球面在直角坐标系中实际上是三维结构，除非距离相对于半径非常之小，否则不能当作平面处理。

两点球面距离计算是天文或地理学中最常用的计算之一，是目标查找、锥形检索、交叉证认等计算中最基础的运算。比如查找距离某个位置最近的山峰，需要先对附近的山峰都计算一遍距离，取出距离最小的一个。而锥形检索是虚拟天文台的数据检索协议之一，用于在球面上查找与某点的距离小于指定半径的目标列表，几乎所有提供了星表查询功能的系统都有类似功能[1]。基于位置的星表交叉证认则可视为是对一个星表A中所有的源做一遍锥形检索取得与其最近的另一星表的源，这样就完成了A、B两个星表的交叉[2]。这些应用的基础都是球面距离计算。

两点球面距离计算，计算的是两点在球面上的最短距离，即球心与两点组成的大圆上两点之间较小的那段圆弧的长度，天文上习惯使用该短圆弧对应的大圆圆心角，称为角间距（angular separation）或角距离。球面距离需要使用球面几何进行求解，要大量使用三角函数。而计算机的数值计算精度有限，在对三角函数、反三角函数进行计算时容易出现舍入误差。多个函数的误差积累之后，将导致严重的结果偏离，甚至无法得到正确的结果。前人已经推导了很多数学公式求解球面距离，但在何种情况下应使用哪个公式，公式的精度如何，却鲜少有人讨论。因此，本文考察天文技术中常用的球面距离计算公式，给出它们的算法，并对比它们的精度，为球面距离算法的选择提供参考。

## 1 球面几何方法

球面距离计算可直接通过球面几何推导出来。较常用的计算公式有大圆(Great-circle)公式[1]、Haversine公式[2]等。另外还有Vincenty公式[3]被应用到Astropy等代码库中，但其复杂

---

* 基金项目：国家自然科学基金（11503051）；国家自然科学基金天文联合基金（U1531111, U1531115, U1531246, U1731125, U1731243）；国家科技部国家科技基础条件平台项目"国家地球系统科学数据共享服务平台"、"国家基础科学数据共享服务平台"（DKA2017-12-02-07）资助.

收稿日期：2018-04-08；修订日期：2018-05-10

作者简介：樊东卫，男，博士，研究方向：天文信息技术，Email：fandongwei@nao.cas.cn

1 https://en.wikipedia.org/wiki/Great-circle_distance

度过高，本文在第3.1节中简要提及。

设有两点赤道坐标为$p_1(\alpha_1,\delta_1)$、$p_2(\alpha_2,\delta_2)$，求它们的球面角距离d。其中大圆公式为方程(1)；当两点距离很小时，也有人使用简化的(2) 式 [2,3,4,5,6]；Haversine公式如(3)式。

$$d=\arccos\left[\sin\delta_1\cdot\sin\delta_2+\cos\delta_1\cdot\cos\delta_2\cdot\cos\left(\alpha_1-\alpha_2\right)\right] \tag{1}$$

$$d=\sqrt{\left[\left(\alpha_1-\alpha_2\right)\cdot\cos\left(\frac{\delta_1+\delta_2}{2}\right)\right]^2+\left(\delta_1-\delta_2\right)^2} \tag{2}$$

$$d=2\cdot\arcsin\sqrt{\sin^2\left(\frac{\delta_1-\delta_2}{2}\right)+\cos\delta_1\cdot\cos\delta_2\cdot\sin^2\left(\frac{\alpha_1-\alpha_2}{2}\right)} \tag{3}$$

计算机中通常使用IEEE754二进制浮点数算术标准处理浮点数，其中单精度32位(float或single)真值可精确到小数点后6~9位，双精度64位(double)真值可精确到小数点后15~17位[4]。本文使用C++语言编写程序对3个公式的精度进行了对比，结果如表1。本文使用的单位均为度（°），表中有两行数据时，上方的值为单精度计算的结果，下方值为双精度计算的结果。

**表 1 使用不同的点测试常用球面距离公式的精度（单位：度）**

**Table 1 Accuracy testing at different points for widely used formulas (unit: degree)**

| p1 | p2 | 简化的大圆公式 | 大圆公式 | Haversine |
|---|---|---|---|---|
| 0,0 | 0.01,0 | 0.00999999884516001<br>0.01000000000000000 | 0.00000000000000000<br>0.00999999998265327 | 0.00999999884516001<br>0.01000000000000000 |
| 0,0 | 0,0.01 | 0.00999999884516001<br>0.01000000000000000 | 0.00000000000000000<br>0.00999999998265327 | 0.00999999884516001<br>0.01000000000000000 |
| 180,0 | 180.01,0 | 0.00998573563992977<br>0.00999999999999938 | 0.00000000000000000<br>0.00999999998265327 | 0.00998573563992977<br>0.00999999999999938 |
| 42,43 | 42.01,43 | 0.00731059815734625<br>0.00731353701619125 | 0.00000000000000000<br>0.00731353703719986 | 0.00731059815734625<br>0.00731353701187370 |
| 42,43 | 42,43.01 | 0.00999939627945423<br>0.00999999999999302 | 0.00000000000000000<br>0.01000000001909975 | 0.00999939627945423<br>0.00999999999999302 |
| 0,90 | 180,89.99 | 0.01863496564328671<br>0.01862095887437574 | 0.00000000000000000<br>0.00999999998265327 | 0.01000121701508760<br>0.01000000000000639 |
| 0,0 | 0.1,0 | 0.09999999403953552 | 0.10087054967880249 | 0.09999999403953552 |

[2] https://en.wikipedia.org/wiki/Haversine_formula
[3] https://en.wikipedia.org/wiki/Vincenty%27s_formulae
[4] https://en.wikipedia.org/wiki/Single-precision_floating-point_format

|  |  |  |  |  |
|---|---|---|---|---|
|  |  | 0.10000000000000001 | 0.10000000000019954 | 0.10000000000000001 |

由表1可见，单精度下,两点之间距离较近时，大圆公式返回的是全0，出现了严重的舍入误差,而距离较大时误差也较大。双精度下，大圆公式能够返回结果，但比Haversine公式的精度要差。大圆公式的简化形式多数时候与Haversine公式的精度相当。这3个公式共有的一个问题，它们均有可能略微超过准确值，当严格限定一个边界值的时候，可能导致漏源。因而在实际应用时，需要考虑将边界值略微外扩，以将因为计算精度而漏掉的源包含在内。

值得注意的是，大圆公式的简化形式在极点附近时误差非常大。虽然它在上述公式里是计算复杂度最低的，并且在两极之外、距离较小时精度与Haversine相近，但是在实际使用该公式时仍应当非常小心。

## 2 直角坐标系方法

在基于赤道坐标系进行求解之外，也可以使用直角坐标系，通过三维坐标向量对两点球面距离进行计算。可视天球半径为单位1，以天球球心为三维直角坐标系原点，赤道坐标系北天极点方向为$z$轴方向，天赤道面上赤经为0点的方向为$x$正向，遵循右手坐标系与$x$-$z$相交的轴为$y$轴方向。从而可将赤经、赤纬转换为三维直角坐标($x, y, z$)，其转换关系如（4-6）式[7]。

$$x = \cos(\delta)\cdot\cos(\alpha) \tag{4}$$

$$y = \cos(\delta)\cdot\sin(\alpha) \tag{5}$$

$$z = \sin(\delta) \tag{6}$$

这样两点距离计算可以直接使用（8）式进行计算。该公式可简单地从Haversine公式中推得，其中Haversine公式根号中的部分（7式）实际上是两点的三维空间直线距离的一半的平方。

$$\begin{aligned} A &= \sin^2\left(\frac{\delta_1-\delta_2}{2}\right)+\cos\delta_1\cdot\cos\delta_2\cdot\sin^2\left(\frac{\alpha_1-\alpha_2}{2}\right) \\ &= \frac{1}{2^2}\cdot\left[(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2\right] \\ &= \sin^2\left(\frac{d}{2}\right) \end{aligned} \tag{7}$$

$$d = 2\cdot\arcsin\sqrt{A} = 2\cdot\arcsin\left[\frac{1}{2}\cdot\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}\right] \tag{8}$$

直角坐标系的计算精度方面，本文依旧使用C++语言在单精度及双精度两个环境下对公式进行计算，并与Haversine的结果对比，结果如表2。从表2可以看出，直角坐标系方法的精度与Haversine几乎完全一致，而直角坐标系方法有时甚至优于Haversine（见42,43两行数据）。

表 2 Haversine 与直角坐标系计算结果对比（单位：度）

Tab. 2 Result comparison between Haversine and Cartesian method (unit: degree)

| p1 | p2 | Haversine | 直角坐标系方法 |
|---|---|---|---|
| 0,0 | 0.01,0 | 0.00999999884516001<br>0.01000000000000000 | 0.00999999884516001<br>0.01000000000000000 |
| 0,0 | 0,0.01 | 0.00999999884516001 | 0.00999999884516001 |

| | | | |
|---|---|---|---|
| | | 0.01000000000000000 | 0.01000000000000000 |
| 180,0 | 180.01,0 | 0.00998573563992977<br>0.00999999999999938 | 0.00998573563992977<br>0.00999999999999938 |
| 42,43 | 42.01,43 | 0.00731059815734625<br>0.00731353701187370 | 0.00731242587789893<br>0.00731353701187507 |
| 42,43 | 42,43.01 | 0.00999939627945423<br>0.00999999999999302 | 0.00999987311661243<br>0.00999999999999879 |
| 0,90 | 180,89.99 | 0.01000121701508760<br>0.01000000000000639 | 0.01000121701508760<br>0.01000000000000639 |
| 0,0 | 0.1,0 | 0.09999999403953552<br>0.10000000000000001 | 0.09999999403953552<br>0.10000000000000001 |

直角坐标系方法在进行距离比较时还可以进一步进行优化，如（9）式：设两点直角坐标为$p_1(x_1,y_1,z_1)$、$p_2(x_2,y_2,z_2)$。其优点是可以预先计算x、y、z与threshold（设最大角距离为$\theta$），从而避免在实际进行距离计算时使用非常耗时的三角函数，而只需要三次减法、三次乘法、两次加法、一次比较，大大减少了计算量，非常适用于大规模数据计算。相应的，它的缺点是需要额外占用存储空间来保存x、y、z三个值，在对内存占用要求较苛刻的环境中可能会成为问题。因而，实际的公式选择，仍应视数据规模、计算环境而定。另外，基于三维向量的计算方法，还有法线向量的形式，但其计算过程比本节要复杂，本文将在第3.3节中进行简要描述。

$$2^2 \cdot A = (x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2 \leqslant 2^2 \cdot \sin^2\left(\frac{\theta}{2}\right) = threshold \tag{9}$$

## 3 计算库

许多天文软件包也包含了球面距离计算功能。本部分选择了几个安装、使用较便捷，应用也比较广泛的库，演示了其使用方法，并进行测试。由于难以分辨软件包中实际使用的是单精度还是双精度，此部分将不再对此进行区分，而直接与Haversine双精度结果进行比较。

### 3.1 Astropy

当前天文界已经非常流行使用Python来进行数据处理，其中Astropy[8]为Python的推广传播功不可没。Astropy进行球面距离计算需要使用astropy.units及astropy.coordinates.SkyCoord计算。调用方法如下，其中$d$为两点之间的角距离（单位是度）：

$c_1$ = SkyCoord(ra$_1$*units.rad,dec$_1$*units.rad)

$c_2$ = SkyCoord(ra$_2$*units.rad,dec$_2$*units.rad)

$d$ = $c_1$.separation($c_2$).deg

为公平起见，在64位Windows 10中的64位python3.6.3重新实现了Haversine算法，并与Astropy的距离计算函数进行了对比。对比结果见表3，可以看到两种方法基本一致，不分伯仲。

表 3 Windows 64 位 Python3 环境下 Haversine 与 Astropy 计算结果对比（单位：度）

Tab. 3 Result comparison between Haversine and Astropy on Windows 64bit Python3 (unit: degree)

| p1 | P2 | Haversine | Astropy |
|---|---|---|---|
| 0,0 | 0.01,0 | 0.01 | 0.01 |
| 0,0 | 0,0.01 | 0.01 | 0.01 |
| 180,0 | 180.01,0 | 0.009999999999999377 | 0.009999999999990905 |
| 42,43 | 42.01,43 | 0.007313537011873697 | 0.007313537011872698 |
| 42,43 | 42,43.01 | 0.009999999999993016 | 0.009999999999994572 |
| 0,90 | 180,89.99 | 0.010000000000006394 | 0.010000000000006394 |
| 0,0 | 0.1,0 | 0.1 | 0.1 |

阅读Astropy的源代码可发现其距离计算函数separation实际调用的是astropy/coordinates/angle_utilities.py[5]中的angular_separation(lon1, lat1, lon2, lat2)函数，其代码实现使用Vincenty公式，即（10）式。该式在所有位置的距离计算都更稳定，但计算复杂度也更高[9]。从上述计算对比结果来看，Astropy并没有处处都比Haversine的精度更高，如（180,0）一行；当然，Astropy在(42,43)上的精度也比Haversine更好。但是两者的差距都非常小，几乎可以忽略不计。从比较结果上看，Haversine已经相当精确。当然，若是对计算稳定性有更高要求，可以考虑Vincenty公式。

$$d = \arctan \frac{\sqrt{[\cos\delta_2 \cdot \sin(\alpha_1-\alpha_2)]^2 + [\cos\delta_1 \cdot \sin\delta_2 - \sin\delta_1 \cdot \cos\delta_2 \cdot \cos(\alpha_1-\alpha_2)]^2}}{\sin\delta_1 \cdot \sin\delta_2 + \cos\delta_1 \cdot \cos\delta_2 \cdot \cos(\alpha_1-\alpha_2)} \quad (10)$$

## 3.2 HTM

分层三角网格（Hierarchical Triangular Mesh, HTM）[10]是一种对球面的多层次、递归三角分割方法。分层三角网格目前公开提供下载的软件包[6]中包含了C#代码库及一个SQL Server扩展包。其中Spherical.Htm.Sql.cs文件中包含了函数fDistanceEq(ra1,dec1,ra2,dec2)，可用于计算距离。需要注意的是，fDistanceEq的返回值单位是arcmin，不是degree。针对分层三角网格，本文使用C#实现了Haversine函数并与SphericalHTM v3.1.2的fDistanceEq函数进行对比。如表4，其结果基本相同。查看源代码发现fDistanceEq实际上也使用了Haversine公式进行计算，只是多了degree转arcmin的转换。

除了C#程序中可以直接调用 fDistanceEq外，HTM软件包提供的SQL Server数据库扩展包中也带有此函数，在数据库中使用非常方便。

[5] 基于 Astropy 2.0.2，参考网址 https://github.com/astropy/astropy/blob/master/astropy/coordinates/angle_utilities.py

[6] http://www.skyserver.org/htm/index.html

表 4 C#环境下 Haversine 与 HTM fDistanceEq 计算结果对比（单位：度）

Table 4 Result comparison between Haversine and HTM fDistanceEq on C# (unit: degree)

| p1 | p2 | Haversine | HTM fDistanceEq |
|---|---|---|---|
| 0,0 | 0.01,0 | 0.01 | 0.01 |
| 0,0 | 0,0.01 | 0.01 | 0.01 |
| 180,0 | 180.01,0 | 0.00999999999999938 | 0.00999999999999938 |
| 42,43 | 42.01,43 | 0.0073135370118737 | 0.00731353701187507 |
| 42,43 | 42,43.01 | 0.00999999999999302 | 0.00999999999999879 |
| 0,90 | 180,89.99 | 0.0100000000000064 | 0.0100000000000064 |
| 0,0 | 0.1,0 | 0.1 | 0.1 |

## 3.3 SLALIB与SOFA

SLALIB即Starlink library of positional astronomy routines[7]，使用Fortran 77进行实现，其目标是让天文工作者更简便快捷地编写精确且可靠的天体测量程序。虽然它是由Fortran 77实现的，但是已经有人为其编写了Python接口pyslalib[8]，因而可以在已有的Python环境中获得Fortran 77的计算精度。

Pyslalib中的slalib.sla_dsep直接对应了SLALIB中的sla_DSEP (A1, B1, A2, B2)球面距离计算函数，在64位的Ubuntu 16.04.4操作系统中使用64位 Python3.5.2环境将其与Haversine函数的计算结果进行了对比，如表5。首先可以看到Linux版的Python Harversine程序计算结果与Windows一致，表明了Python3在不同的操作系统中的表现一致。其次pyslalib的计算结果，除了(42,43)两行有轻微差别，与Haversine完全一致，精确也非常高。查看SLALIB的源代码，可以看到其使用了第2节中的直角坐标系三维向量作为计算单元。但与第2节不同，SLALIB进一步将其转换为法线向量[9]，即$\vec{n}_1$与$\vec{n}_2$，再使用(11)式[11]进行计算。使用向量的主要优势是向量代数可取代部分三角函数的计算，其数值计算稳定性更好，能保持较好的精度，并且在边界点（如南北天极、0-360度边界）也无异常。

天体测量中常用的SOFA库[10]的球面计算函数iauSeps[11]也同样使用了(11)式。SOFA即Standards of Fundamental Astronomy，是IAU建立并维护的一个权威的基本天文学标准库，包含有ANSI C及Fotran 77两个版本。表5的最右侧列出了双精度位C++程序调用SOFA计算的结果。

$$d = \arctan\left(\frac{|\vec{n}_1 \times \vec{n}_2|}{\vec{n}_1 \cdot \vec{n}_2}\right) \qquad (11)$$

7 http://stsdas.stsci.edu/cgi-bin/gethelp.cgi?slalib.sys
8 https://github.com/scottransom/pyslalib
9 https://en.wikipedia.org/wiki/N-vector
10 http://www.iausofa.org
11 http://www.iausofa.org/2018_0130_C/SeparationPA.html

表 5 Linux 64 位 Python3 环境下 Haversine 与 pyslalib 及双精度 C++版本 SOFA 计算结果对比（单位：度）

Table 5 Result comparison among Haversine and pyslalib on Linux 64bit Python3 and double precision C++ version SOFA (unit: degree)

| p1 | P2 | Haversine | pyslalib | SOFA |
|---|---|---|---|---|
| 0,0 | 0.01,0 | 0.01 | 0.01 | 0.01000000000000000 |
| 0,0 | 0,0.01 | 0.01 | 0.01 | 0.01000000000000000 |
| 180,0 | 180.01,0 | 0.009999999999999377 | 0.009999999999999377 | 0.00999999999999938 |
| 42,43 | 42.01,43 | 0.007313537011873697 | 0.007313537011875421 | 0.00731353701187542 |
| 42,43 | 42,43.01 | 0.009999999999993016 | 0.0099999999999971 | 0.00999999999999710 |
| 0,90 | 180,89.99 | 0.010000000000006394 | 0.010000000000006394 | 0.01000000000000639 |
| 0,0 | 0.1,0 | 0.1 | 0.1 | 0.10000000000000001 |

## 4 数据库

关系型数据库均支持SQL语句，因而可以在不同的系统中直接使用SQL语句实现Haversine公式进行距离计算。由于每种数据库使用的数学函数、计算精度略有差别，本文在Microsoft SQL Server 2008、PostgreSQL 9.4.5、MySQL 5.7.18上进行了测试。测试结果结果如表6，从表6可以看到，3个数据库的差距不大，MySQL默认保留的精度更多一些。

表 6 不同数据库中使用 Haversine 公式计算距离的精度对比（单位：度）

Table 6 Accuracy testing for pure SQL Haversine on different database (unit: degree)

| p1 | p2 | MS SQL Server | PostgreSQL | MySQL |
|---|---|---|---|---|
| 0,0 | 0.01,0 | 0.00999999999999926 | 0.01 | 0.01 |
| 0,0 | 0,0.01 | 0.00999999999999926 | 0.01 | 0.01 |
| 180,0 | 180.01,0 | 0.00999999999999811 | 0.00999999999999938 | 0.009999999999999377 |
| 42,43 | 42.01,43 | 0.00731353701187193 | 0.0073135370118737 | 0.007313537011873697 |
| 42,43 | 42,43.01 | 0.00999999999999697 | 0.00999999999999302 | 0.009999999999993016 |
| 0,90 | 180,89.99 | 0.0100000000000109 | 0.0100000000000064 | 0.010000000000006394 |
| 0,0 | 0.1,0 | 0.0999999999999995 | 0.1 | 0.1 |

除了Haversine公式，数据库中使用直角坐标系方法实际上更方便。数据库对I/O进行了

大量优化，可以更好地对数据进行调度。因而，直角坐标系方法虽然需要增加$x$、$y$、$z$ 3个字段，但并不会给数据库增加太大的负担，还可以大大降低计算的复杂度，更快地取得海量数据的计算结果。

## 4.1 PostgreSQL数据库插件

数据库中除了可以直接使用公式之外，也可以使用各种数据库扩展插件进行相关计算，减少使用者编程的麻烦。如PostgreSQL数据库有PostGIS[12]、pgsphere[13]、Q3C[14]、H3C[15]等多个插件支持球面距离计算。其中PostGIS是专为地理信息系统设计的，虽然也可以在天文上使用，但是需要将米、千米等单位转换回弧度，太过繁琐低效。而pgphere、Q3C[12]、H3C则是专门为天文检索设计的，更为简洁。下文演示了3种插件的使用方法，表7对这3个插件的结果进行了对比。需要注意的是Q3C和H3C直接使用度进行计算，而pgsphere需要先转成弧度进行计算，再将结果转回角度。

```
SELECT a ra1,b dec1,c ra2, d dec2,
q3c_dist(a,b,c,d) q3c,
h3c_dist(a,b,c,d) h3c,
DEGREES(spoint(RADIANS(a),RADIANS(b))<->spoint(RADIANS(c),RADIANS(d))) pg
FROM (VALUES (0.0,0.0,0.01,0.0),(0.0,0.0,0.0,0.01),(180.0,0.0,180.01,0.0),(42.0,43.0,42.01,43.0),(42.0,43.0,42.0,43.01),(0.0,90.0,180.0,89.99),(0.0,0.0,0.1,0.0)) AS v(a,b,c,d)
```

表 7 PostgreSQL 数据库 Q3C、H3C、pgsphere 插件的球面距离计算精度比较（单位：度）

Table 7 Accuracy testing for different PostgreSQL extensions (unit: degree)

| p1 | p2 | Q3C | H3C | pgsphere |
|---|---|---|---|---|
| 0.0,0.0 | 0.01,0.0 | 0.01 | 0.01 | 0.00999999998265327 |
| 0.0,0.0 | 0.0,0.01 | 0.01 | 0.01 | 0.00999999998265327 |
| 180.0,0.0 | 180.01,0.0 | 0.00999999999999091 | 0.00999999999999938 | 0.00999999998265327 |
| 42.0,43.0 | 42.01,43.0 | 0.0073135370118727 | 0.0073135370118737 | 0.00731353703719986 |
| 42.0,43.0 | 42.0,43.01 | 0.00999999999999801 | 0.00999999999999302 | 0.0100000000190997 |
| 0.0,90.0 | 180.0,89.99 | 0.0100000000000064 | 0.0100000000000064 | 0.00999999998265327 |
| 0.0,0.0 | 0.1,0.0 | 0.1 | 0.1 | 0.1000000000002 |

从三者的结果来看，精度大抵相当，没有太大差距，在极点的计算也都没有太大的偏差。经阅读源代码，可以看到h3c_dist函数使用了Haversine公式；q3c_dist函数使用了Haversine的一种变体；pgsphere的情况则较为复杂，它在距离大时使用了大圆公式，距离

12 https://postgis.net/
13 https://github.com/akorotkov/pgsphere
14 https://github.com/segasai/q3c
15 http://cds.u-strasbg.fr/resources/doku.php?id=h3c

较小时使用直角坐标公式。从计算结果来看，这3个插件都不失为成熟可用的扩展，具体要使用哪个，还要看没有其他的需求。如Q3C、H3C均提供了join函数（分别是q3c_join、h3c_join），对两星表交叉证认进行了优化。而pgsphere提供了丰富的球面几何计算，如球面形状的面积计算，球面形状的交、并计算等。

## 5 总结

本文探讨了大圆公式、简化的大圆公式、Haversine公式等球面距离计算方法，并对比了它们在计算机中进行单精度及双精度数值计算的结果。在此基础上，引申出了基于三维直角坐标系的距离计算方法。结果表明大圆公式在单精度下舍入误差非常严重，而大圆公式的简化公式在两极或两点距离较大时误差很大。Haversine公式与直角坐标系方法的精度相近、都非常高，而直角坐标系方法的计算量较小，适合大数据的计算。此外，还需要注意所有这些公式在计算时，有时候计算结果会略微大于准确值。因而在实际应用时，可能需要将边界值取得略大一点，以将因为舍入误差漏掉的源包含在内。

已有软件包如Astropy、HTM、SLALIB、SOFA以及部分数据库扩展也将球面距离计算模块包含在内。它们所使用的计算方法不尽相同，结果精度也非常高。当需要进行球面距离计算时，可以直接使用这些库，而无须自行实现过于复杂的公式，以免重复工作且易因考虑不周而导致出错。

# Research on Spherical Distance Computation and Accuracy Comparison

Fan Dongwei, He Boliang, Li Changhua, Han Jun, Xu Yunfei, Cui Chenzhou

(National Astronomical Observatories, Chinese Academy of Sciences, Beijing 100101, China)

**Abstract:** Spherical distance (Angular separation) calculation is commonly using in Astronomy and Geography. It is the foundation of object detecting, data query and cross-matching. The spherical distance can be computed by Spherical Geometry method, and people have deduced many formulas. But the precision of computer is limited, due to rounding error in the numerical computation. This article will inspect several widely used formulas, test and compare their results, and discuss their advantage and disadvantage. In addition, this article will demonstrate how to do distance calculation on several astronomical packages and databases. The purpose of this article is helping astronomers to find the suitable method to do their calculation.

**Key words:** Spherical Distance; Angular Separation; Numerical Computation; Calculation Accuracy

参考文献:

［1］ZHANG Hailong, YE Xinchen, LI Huijuan et al. Astronomical Data Query and Release Review [J](张海龙,冶鑫晨,李慧娟,王杰,王万琼,托乎提努尔,聂俊,崔辰州,刘梁,谌俊毅,陈肖,薛岩松,何勃亮,李长华,赵青,肖健,樊东卫,曹子皇,李珊珊,米琳莹,杨哲睿.天文数据检索与发布综述[J].天文研究与技术),2017,14(02):212-228.

［2］ZHANG Hailong, NIE Jun, ZHAO Qing et al. Xinjiang Astronomical Observatory Data Center Custom Uploading Crossmatcher[J](张海龙,聂俊,赵青,冶鑫晨,王杰.新疆天文台在线交叉证认服务[J].天文研究与技术),2017,14(03):347-355.

［3］ ZHANG Yan-xia. Research on Automatic Classification Methods in the Multi-wavelength Astrophysics. Ph.D. Thesis(张彦霞. 多波段天体物理中的自动分类方法研究. 博士学位论文),2003

[4] GAO Dan. Development of Massive Astronomy Data Federation System and Research of Data Mining Algorithms. Ph.D. Thesis(高丹. 海量天文数据融合系统的开发与数据挖掘算法的研究. 博士学位论文), 2008

[5] ZHAO Qing. Research on High-Efficient Massive Data Oriented Astronomical Cross-Match. Ph.D. Thesis(赵青. 面向海量数据的高效天文交叉证认的研究. 博士学位论文),2010

[6] DU Peng. The Cross-match and Aggregation of Large-scale Astronomical Catalogs Data. M.D. Thesis(杜鹏. 海量天文星表数据的交叉与融合), 2013

[7] Gray J, Nieto-Santisteban M A, Szalay A S. The zones algorithm for finding points-near-a-point or cross-matching spatial datasets[J]. arXiv preprint cs/0701171, 2007.

[8] Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., ... & Conley, A. Astropy: A community Python package for astronomy. Astronomy & Astrophysics, 2013, 558, A33.

[9] Vincenty, T. Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations. Survey review, 1975, 23(176), 88-93.

[10] Kunszt, P. Z., Szalay, A. S., & Thakar, A. R. The hierarchical triangular mesh. In Mining the sky. Springer, Berlin, Heidelberg.2001. pp. 631-637)

[11] Gade, Kenneth. A non-singular horizontal position representation. The Journal of Navigation. Cambridge University Press.2010. Vol.63 (3), pp.395–417

[12] Koposov, S., & Bartunov, O. Q3C, Quad Tree Cube--the new sky-indexing concept for huge astronomical catalogues and its realization for main astronomical queries (cone search and Xmatch) in open source database PostgreSQL. In Astronomical Data Analysis Software and Systems XV, 2006, Vol. 351, p. 735